IA 2

1. **Discuss RMI invocation semantics and tabulate failure handling mechanism for each.**

The RMI invocation semantics are as follows:

Maybe Semantics

With maybe semantics, the remote procedure call may be executed once or not at all.

Maybe semantics arises when no fault-tolerance measures are applied and can suffer from the following types of failures:

Omission failures if the request or result message is lost.

Crash failures when the server containing the remote operation fails.

If the result message has not been received after a timeout and there are no retries, it is uncertain whether the procedure has been executed.

At-least-once Semantics

With at-least-once semantics, the invoker receives wither a result, in which case the invoker knows that the procedure was executed at least once, or an exception informing it that no result was received.

At-least-once semantics can be achieved by the retransmission of request messages.

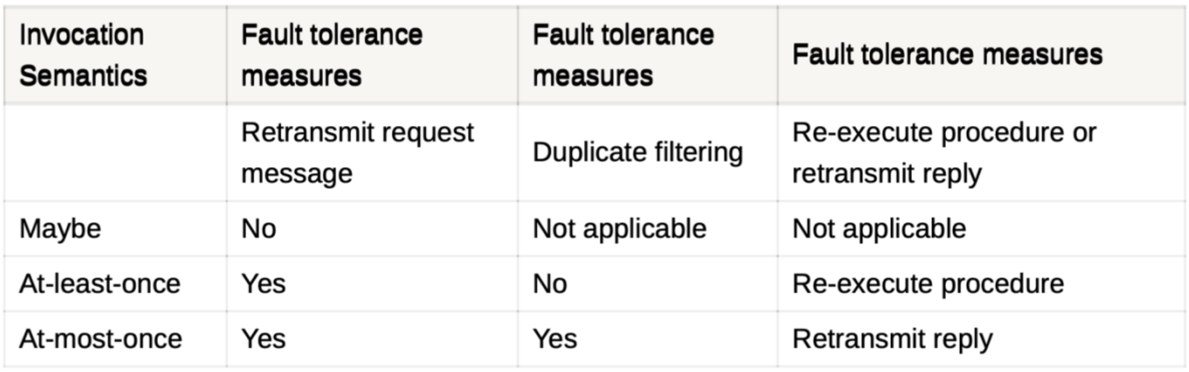
At-lease-once semantics can suffer from the following types of failures: Crash failures when the server containing the remote procedure fails.

Arbitrary failures - in cases when the request message is retransmitted, the remote server may receive it and execute the procedure more than once, possibly causing wrong values to be stored and returned.

At-most-once Semantics

With at-most-once semantics, the caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received, in which case the procedure will have been executed either once or not at all.

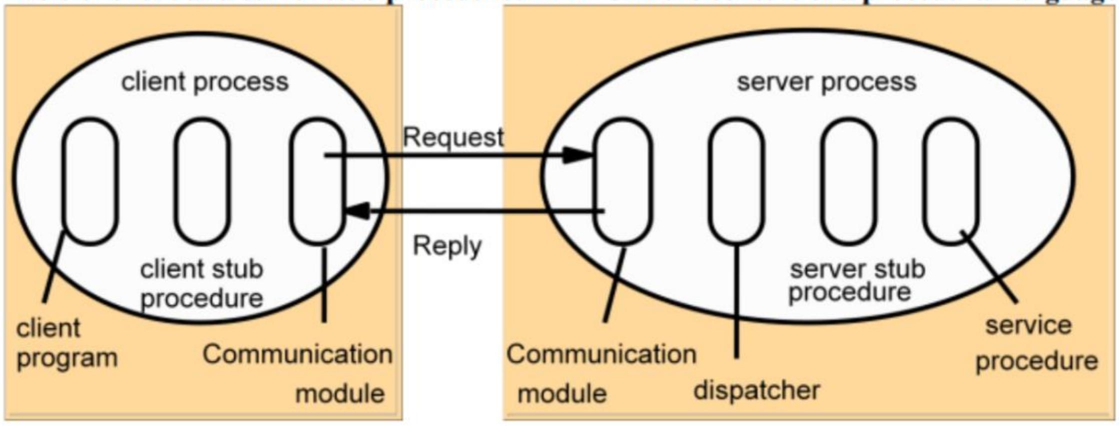
Failure handling mechanism

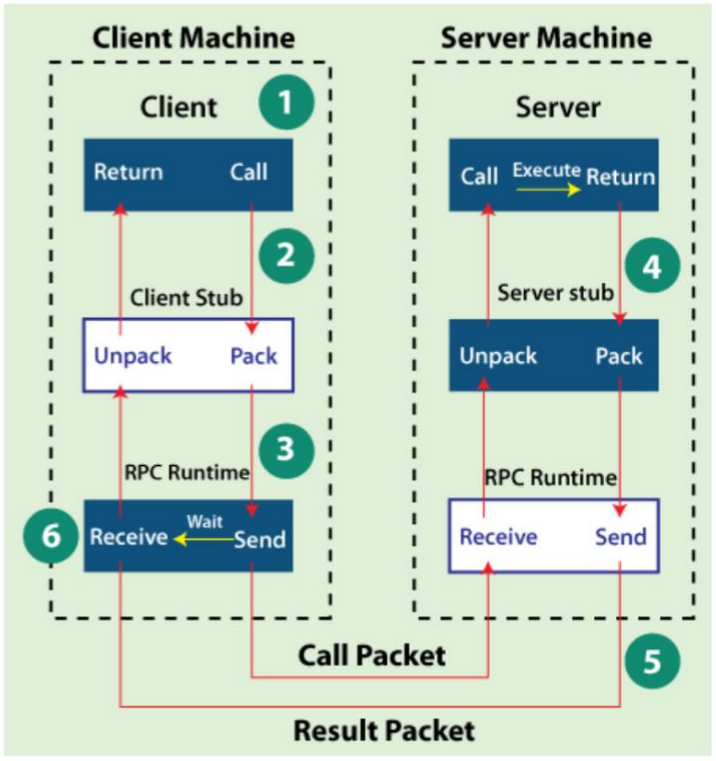


Invocation Semantics	Fault tolerance measures	Fault tolerance measures	Fault tolerance measures
	Retransmit request message	Duplicate filtering	Re-execute procedure or retransmit reply
Maybe	No	Not applicable	Not applicable
At-least-once	Yes	No	Re-execute procedure
At-most-once	Yes	Yes	Retransmit reply

2. **Define RPC and with neat diagram explain its implementation.**

Role of client and server stub procedures in RPC in the context of a procedural language





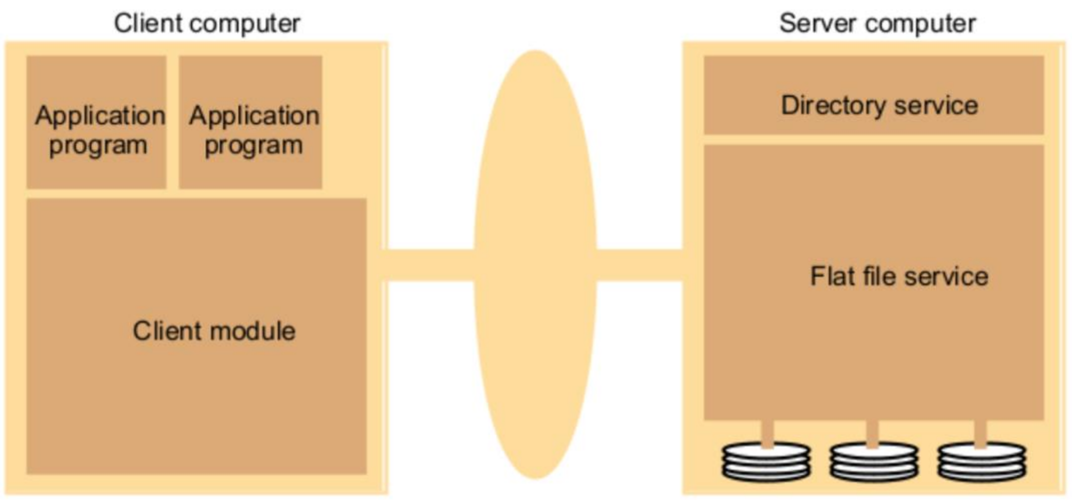
- The Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

- An RPC is analogous to a function call. Like a function call, when RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure.

Implementation

- The software components required to implement RPC are shown in the figure.
- The client calls the client stub. The call is a local procedure call, with parameters pushed onto the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called Marshalling.
- The client's local operating system sends the message from the client machine to the server machine.
- The local operating system on the server machine passes the incoming packets to the server stub.
- The server stub unpacks the parameters from the message. Unpacking the parameters is called Unmarshalling.
- Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

3. **Discuss model architecture of distributed file system and its components.**



File service architecture is an architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components:

Flat File Service:

- The Flat File Service is concerned with the implementation of operations on the contents of the file.
- Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat-file service operations.
- UFIDs are long sequences of bits chosen so that each file has a UFID that is unique among all of the files in a distributed system.
- When the flat file service receives a request to create a file, it generates a new UFID for it and returns the UFID to the requester.

Directory Service:

- The Directory Service provides a mapping between text names for the files and their UFIDs.
- Clients may obtain the UFID of a file by quoting its text name to the directory service.
- The directory service provides the functions needed to generate directories, to add new files to directories, and obtain UFIDs from directories.

Directory files are stored in files of Flat File Services

Client Module:

- A Client Module runs in each client computer, integrating and extending the operations of the flat file service and the directory service under a single application programming interface that is available to user-level programs in client computers.
- It holds information about [redacted] locations of flat-file and directory server processes.
- It helps to achieve better performance through the implementation of a cache of recently used file blocks at the client.

4. **List the various Distributed File Requirements and explain any three in detail.**

Many of the requirements and potential pitfalls in the design of distributed services were first observed in the early development of distributed file systems. Initially, they offered access transparency and location transparency; performance, scalability, concurrency control, fault tolerance and security requirements emerged and were met in subsequent phases of development. We discuss these and related requirements in the following subsections.

Concurrent file updates ◊ Changes to a file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file. This is the well-known issue of concurrency control, discussed in detail in Chapter 12. The need for concurrency control for access to shared data in many applications is widely accepted and techniques are known for its implementation, but they are costly. Most current file services follow modern UNIX standards in providing advisory or mandatory file- or record-level locking.

File replication ◊ In a file service that supports replication, a file may be represented by several copies of its contents at different locations. This has two benefits – it enables multiple servers to share the load of providing a service to clients accessing the same set of files, enhancing the scalability of the service, and it enhances fault tolerance by enabling clients to locate another server that holds a copy of the file when one has failed. Few file services support replication fully, but most support the caching of files or portions of files locally, a limited form of replication. The replication of data is discussed in Chapter 14, which includes a description of the Coda replicated file service.

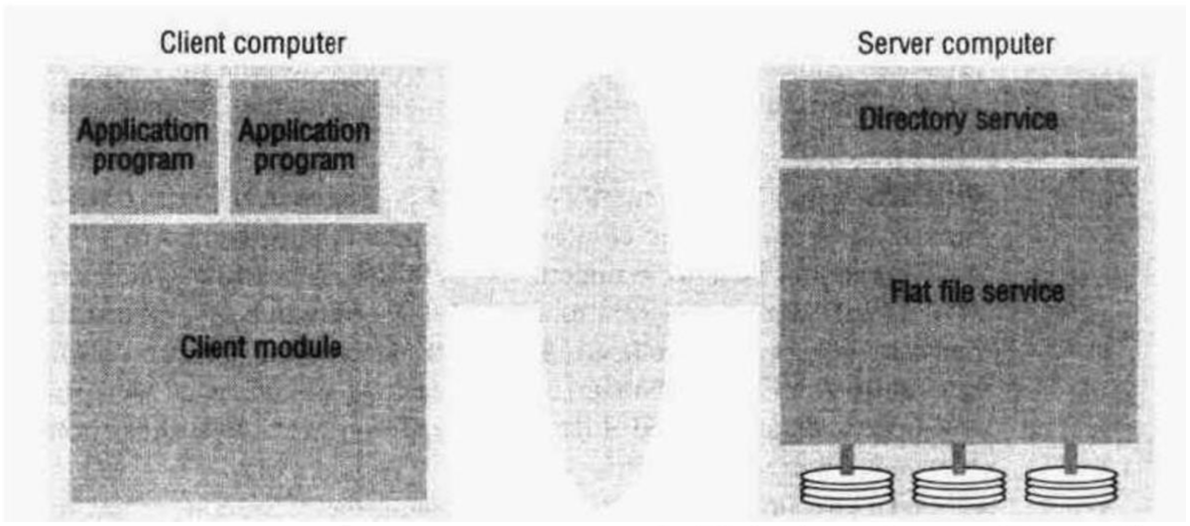
Hardware and operating system heterogeneity ◊ The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers. This requirement is an important aspect of openness.

Consistency ◊ Conventional file systems such as that provided in UNIX offer *one-copy update semantics*. This refers to a model for concurrent access to files in which the file contents seen by all of the processes accessing or updating a given file are those that they would see if only a single copy of the file contents existed. When files are replicated or cached at different sites, there is an inevitable delay in the propagation of modifications made at one site to all of the other sites that hold copies, and this may result in some deviation from one-copy semantics.

Security ◊ Virtually all file systems provide access control mechanisms based on the use of access control lists. In distributed file systems, there is a need to authenticate client requests so that access control at the server is based on correct user identities and to protect the contents of request and reply messages with digital signatures and (optionally) encryption of secret data. We shall discuss the impact of these requirements in our case study descriptions.

5. **With a neat diagram explain the components of file service architecture in brief w.r.t. following**

i. **Flat File Service**
ii. **Directory Service**
iii. **Client Module**



Flat file service ◊ The flat file service is concerned with implementing operations on the contents of files. *Unique file identifiers* (UFIDs) are used to refer to files in all requests for flat file service operations. The division of responsibilities between the file service and the directory service is based upon the use of UFIDs. UFIDs are long sequences of bits chosen so that each file has a UFID that is unique among all of the files in a distributed system. When the flat file service receives a request to create a file, it generates a new UFID for it and returns the UFID to the requester.

Directory service ◊ The directory service provides a mapping between *text names* for files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to the directory service. The directory service provides the functions needed to generate directories, to add new file names to directories and to obtain UFIDs from directories. It is a client of the flat file service; its directory files are stored in files of the flat file service. When a hierarchic file-naming scheme is adopted, as in UNIX, directories hold references to other directories.

Client module ◊ A client module runs in each client computer, integrating and extending the operations of the flat file service and the directory service under a single application programming interface that is available to user-level programs in client computers. For example, in UNIX hosts, a client module would be provided that emulates the full set of UNIX file operations, interpreting UNIX multi-part file names by iterative requests to the directory service. The client module also holds information about the network locations of the flat file server and directory server processes. Finally, the client module can play an important role in achieving satisfactory performance through the implementation of a cache of recently used file blocks at the client.

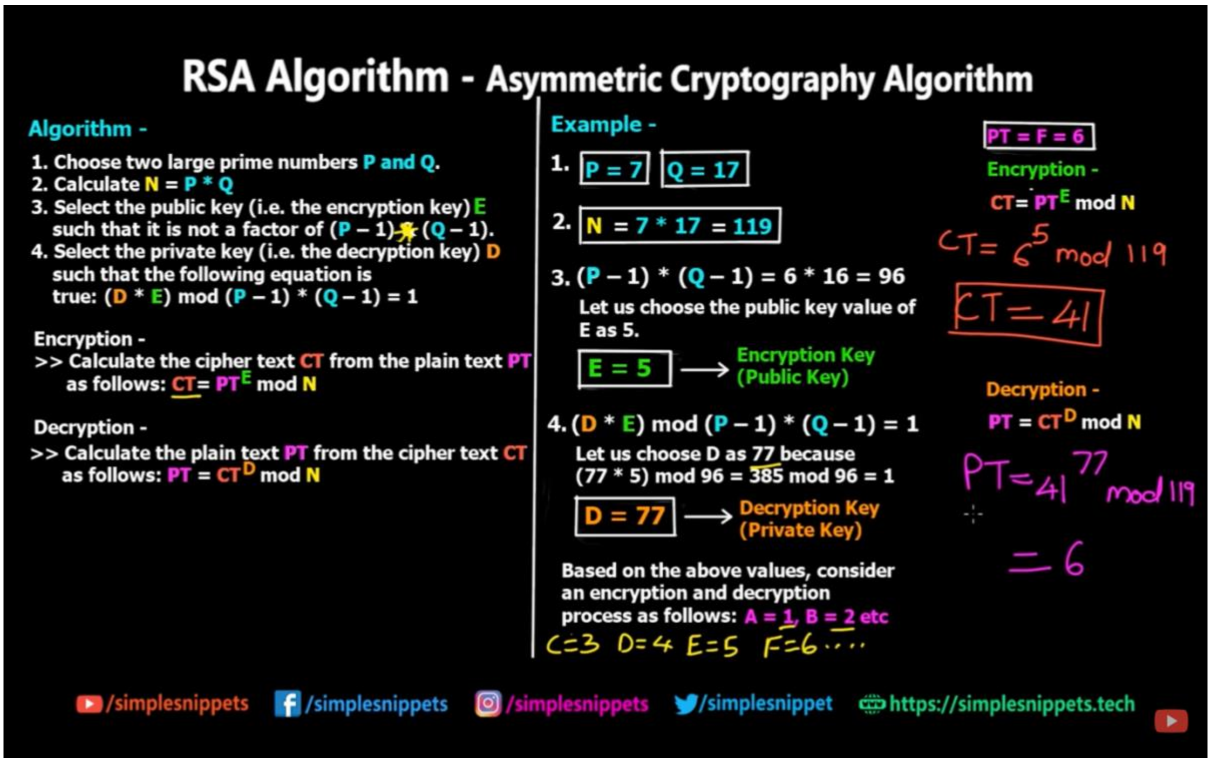
6. **Discuss the distributed file system design requirements.**

Transparency: Some aspects of a Distributed system are hidden from the user. Access: Client programs can be unaware of the distribution of files. The same set of operations is provided for access to remote as well as local files.
Location: The client program should see a uniform namespace.
Mobility: Client programs need not change their tables when files are moved to any other location.
Performance: The client program should continue to perform satisfactorily while the load on the service varies.
Scaling: The service can be expanded to deal with a wide range of load and network sizes.
Concurrent file updates: Changes to the file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file.
File replication: A file may be represented by several copies of its contents at different locations. It has the following benefits:

i. Load balancing to enhance the scalability of the service.
ii. Enhances the fault tolerance.

Hardware and OS heterogeneity: The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers.
Fault tolerance: To cope with transient communication failures, the design can be based on at-most-once invocation semantics.
Consistency: Maintaining the consistency between multiple copies of files.
Security: In distributed file systems, there is a need to authenticate client requests so that access control at the server is based on correct user identities and to protect the contents of the request and reply messages.

7. **Write the steps of RSA Algorithm. Illustrate with an example given Message = 8, P=3 & Q=11.**



RSA Algorithm - Asymmetric Cryptography Algorithm

Algorithm -

1. Choose two large prime numbers P and Q.
2. Calculate $N = P * Q$
3. Select the public key (i.e. the encryption key) E such that it is not a factor of $(P - 1) * (Q - 1)$.
4. Select the private key (i.e. the decryption key) D such that the following equation is true: $(D * E) \bmod (P - 1) * (Q - 1) = 1$

Encryption -

>> Calculate the cipher text CT from the plain text PT as follows: $CT = PT^{E} \bmod N$

Decryption -

>> Calculate the plain text PT from the cipher text CT as follows: $PT = CT^{D} \bmod N$

Example -

1. $\boxed{P = 7}$ $\boxed{Q = 17}$
2. $\boxed{N = 7 * 17 = 119}$
3. $(P - 1) * (Q - 1) = 6 * 16 = 96$

   Let us choose the public key value of E as 5.

   $\boxed{E = 5} \longrightarrow$ Encryption Key (Public Key)
4. $(D * E) \bmod (P - 1) * (Q - 1) = 1$

   Let us choose D as 77 because $(77 * 5) \bmod 96 = 385 \bmod 96 = 1$

   $\boxed{D = 77} \longrightarrow$ Decryption Key (Private Key)

Based on the above values, consider an encryption and decryption process as follows: A = 1, B = 2 etc

C=3 D=4 E=5 F=6....

$\boxed{PT = F = 6}$

Encryption -

$CT = PT^{E} \bmod N$

$CT = 6^{5} \bmod 119$

$\boxed{CT = 41}$

Decryption -

$PT = CT^{D} \bmod N$

$PT = 41^{77} \bmod 119$

$= 6$

/simplesnippets /simplesnippets /simplesnippets /simplesnippet https://simplesnippets.tech

8. **Analyze the following uses of Cryptography with suitable scenarios.**
    i. **Secrecy and integrity**
    ii. **Authentication**

Secrecy and Integrity

- Cryptography is used to maintain the secrecy and integrity of information whenever it is exposed to potential attacks.
- It maintains the secrecy of the encrypted message as long as the decryption key is not compromised. (found)
- It also maintains the integrity of the encrypted information, provided that some redundant information such as a checksum is included and checked.

Scenario:

Secret communication with a shared secret key:

Alice wishes to send some information secretly to Bob. Alice and Bob share a secret key KAB.

- Alice uses KAB and an agreed encryption function E(KAB, M) to encrypt and send any number of messages {Mi}KAB to Bob.
- Bob decrypts the encrypted messages using the corresponding decryption function D(KAB, M).

Authentication

- Cryptography is used in support of mechanisms for authenticating communication between pairs of principals.
- Key is known only to two parties.

Scenario:

Bob has a public/private key pair <KBpub, KBpriv>

- Alice obtains a certificate that was signed by a trusted authority stating Bob's public key Kbpub.
- Alice creates a new shared key KAB, encrypts it using Kbpub using a public-key algorithm, and sends the result to Bob.
- Bob uses the corresponding private key KBpriv to decrypt it.

9. **Discuss asymmetric (public/private key pair-based) cryptography technique and how it can be used in supporting security in distributed systems.**

When a public/private key pair is used, one-way functions are exploited in another way. The feasibility of a public-key scheme was proposed as a cryptographic method that eliminates the need for trust between communicating parties. The basis for all public-key schemes is the existence of a trap-door function. A trap-door function is a one-way function with a secret exit- it is easy to compute in one direction but infeasible to compute its reverse unless the secret is known.

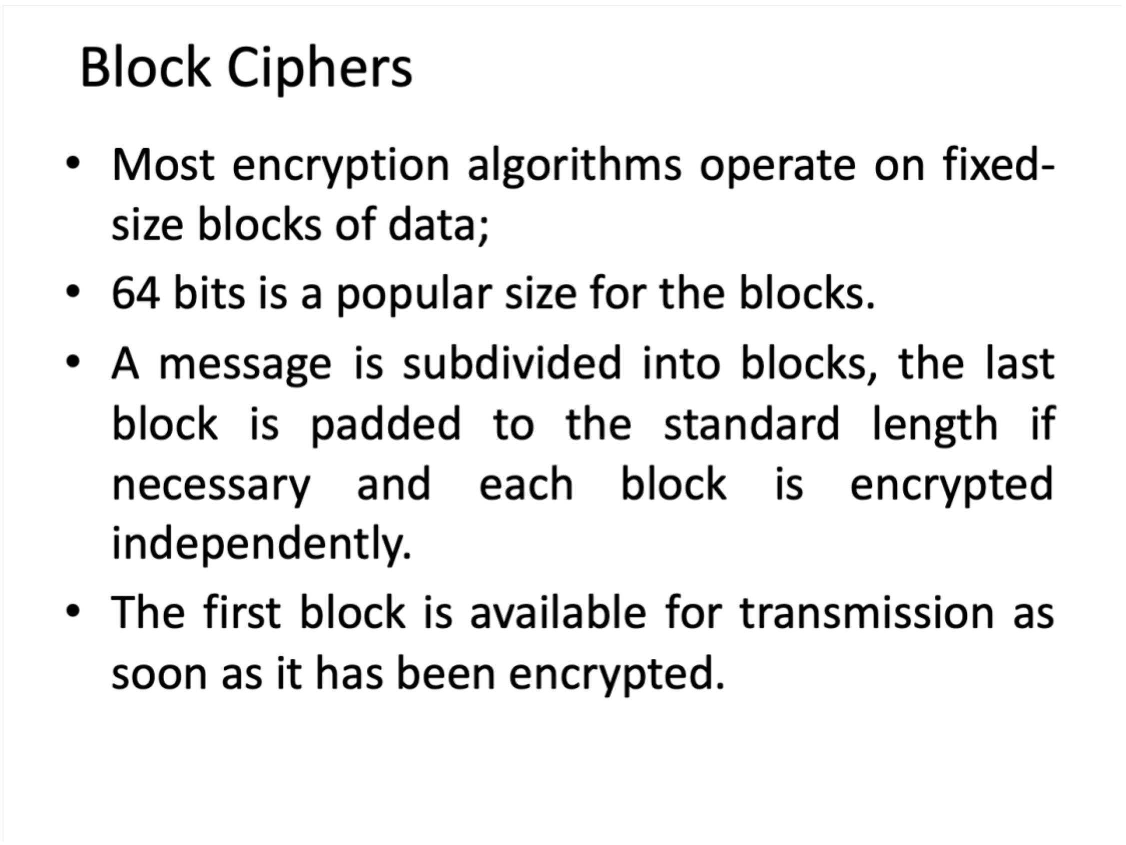
The pair of keys needed for asymmetric algorithms are derived from a common root. The derivation of the pair of keys from the root is a one-way function. In the case of the RSA algorithm, the large numbers are multiplied together, this computation takes only a few seconds, even for very large primes used. The resulting product, N, is computationally infeasible to derive the original multiplicands.

One of the pair of keys is used for encryption. For RSA the encryption procedure obscures the plaintext by treating each block as a binary number and raising it to the power of key, modulo N. The resulting number is the corresponding ciphertext block. The size of N and at least one of the pair of keys is much larger than the safe key size for symmetric keys to ensure that N is not factorizable. For this reason, the potential attacks on RSA are small, its resistance to attacks depends on the infeasibility of factorizing N.

10. **Explain following symmetric key encryption techniques.**

i. Block cipher

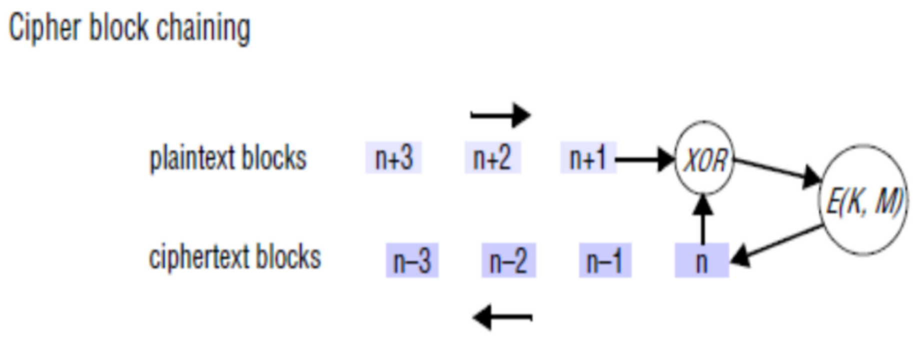
ii. Stream cipher



Block Ciphers

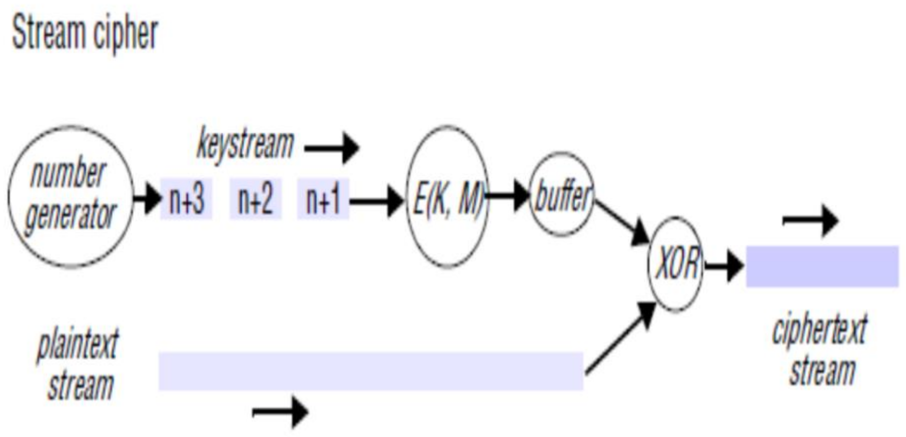
- Most encryption algorithms operate on fixed-size blocks of data;
- 64 bits is a popular size for the blocks.
- A message is subdivided into blocks, the last block is padded to the standard length if necessary and each block is encrypted independently.
- The first block is available for transmission as soon as it has been encrypted.

Cipher block chaining



Stream ciphers

- Stream ciphers are encryption algorithms that can perform encryption incrementally, converting plain text to cipher text one bit at a time.



11. **Write a note on digital signature?**

- Cryptography is used to implement a mechanism known as a digital signature.
- This is similar to the conventional signature, verifying to a third party that a message or a document is an unaltered copy of one produced by the signer.
- This can be achieved by encrypting the message called a digest – using a key that is known only to the signer.
- A digest is a fixed-length value computed by applying a secure digest function.
- The resulting encrypted digest acts as a signature that accompanies the message.
- The originator generates a signature with their private key, and the signature can be decrypted by any recipient using the corresponding public key.